A Short Guide to Using JAGUAR

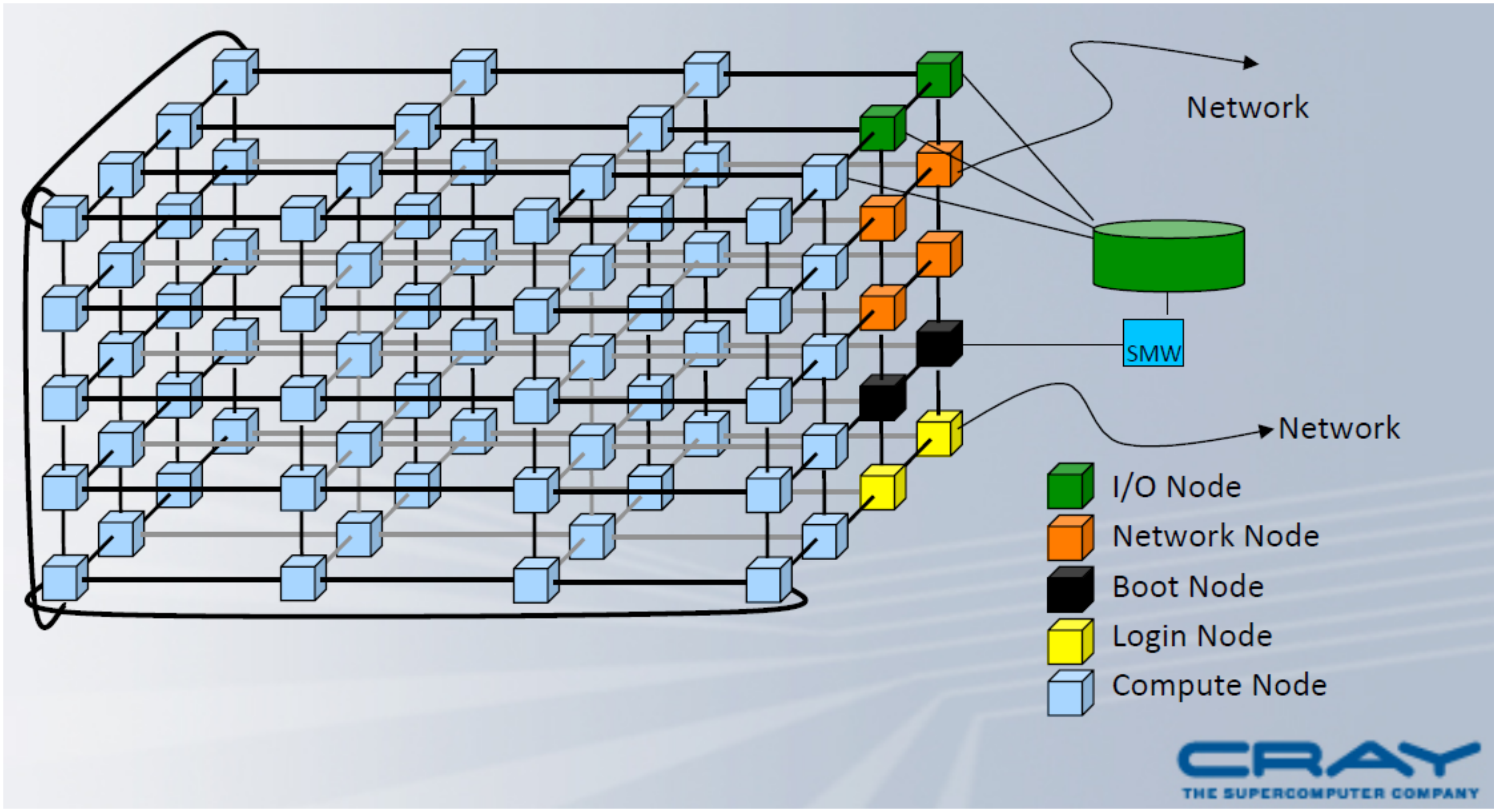
OAK RIDGE LEADERSHIP COMPUTING FACILITY

U.S. DEPARTMENT OF ENERGY

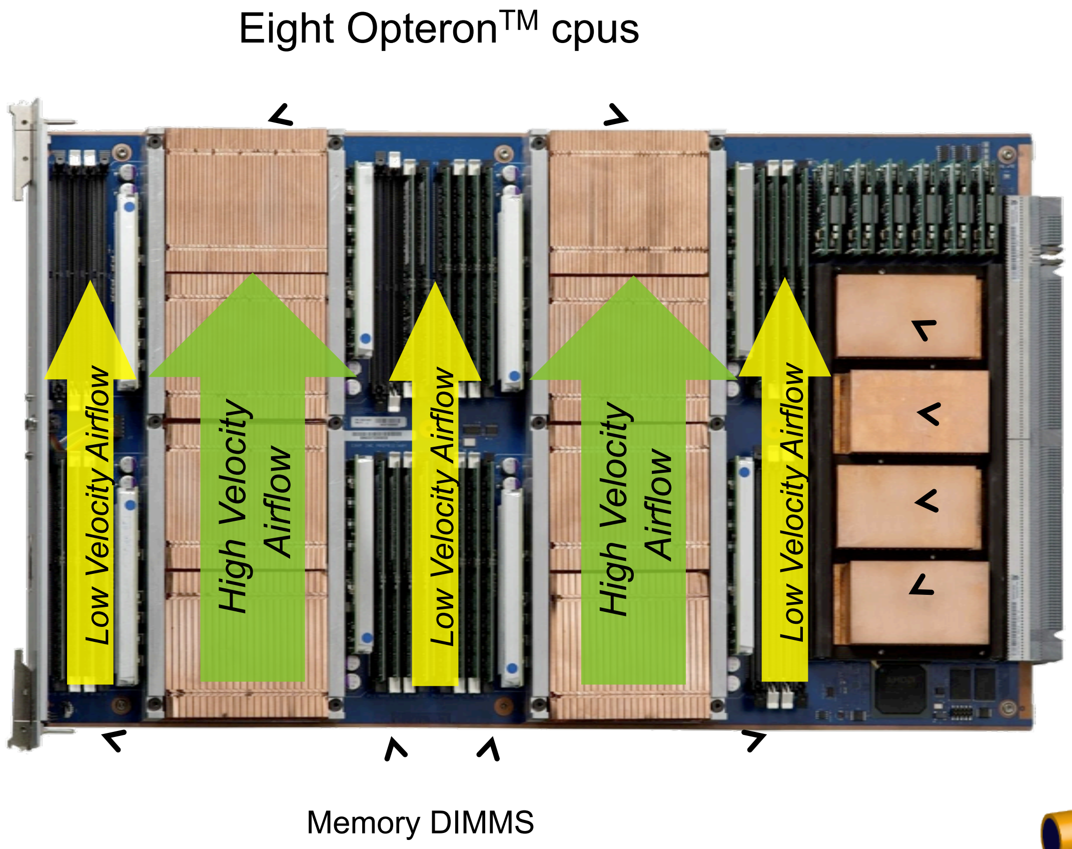
OAK RIDGE NATIONAL LABORATORY
MANAGED BY UT-BATTELLE FOR THE DEPARTMENT OF ENERGY

Jaguar System Overview: System Hardware

- XT System Torus Architecture

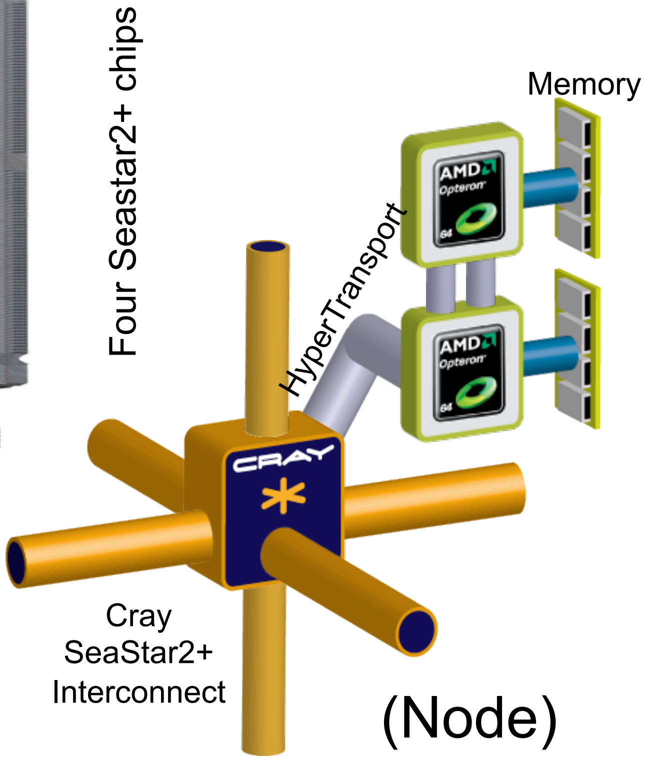


Cray XT5 Blade and Compute Node



Four Compute Nodes per Blade
(2 cpus per node)

Seastar2+ chips form 3-D torus interconnect



(Node)

Six-Core AMD Opteron™ Processor

Performance
- Six-Core AMD Opteron™ Processor
  - 6M Shared L3 Cache
  - North Bridge enhancements (PF + prefetch)
  - 45nm Process Technology
- DDR2-800 Memory
- HyperTransport-3 @ 4.8 GT/sec

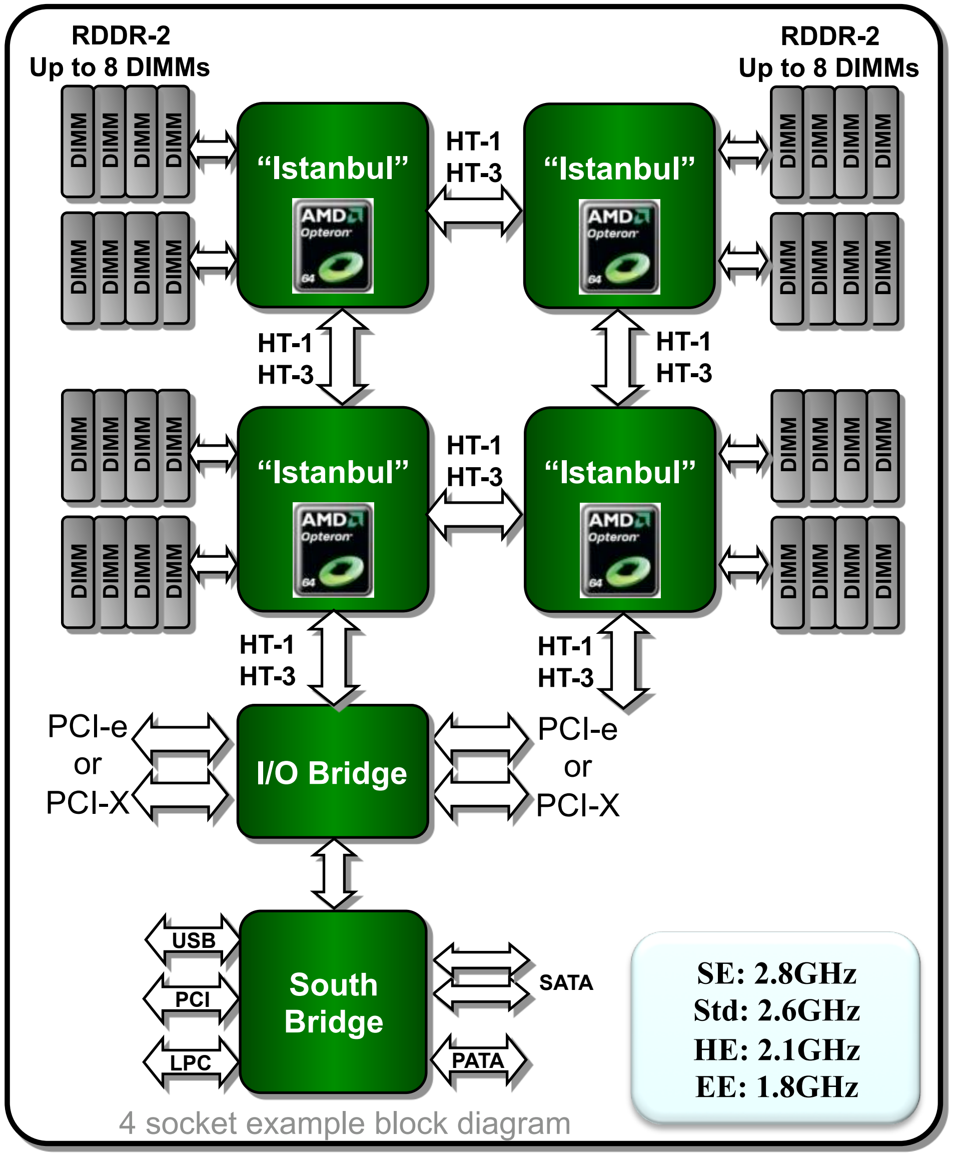
Reliability/Availability
- L3 Cache Index Disable
- HyperTransport Retry (HT-3 Mode)
- x8 ECC (Supports x4 Chipkill in unganged mode)

Virtualization
- AMD-V™ with Rapid Virtualization Indexing

Manageability
- APML Management Link*

Scalability
- 48-bit Physical Addressing (256TB)
- HT Assist (Cache Probe Filter)

Continued Platform Compatibility
- Nvidia/Broadcom-based F/1207 platforms



SE: 2.8GHz
Std: 2.6GHz
HE: 2.1GHz
EE: 1.8GHz

4 socket example block diagram

Logging into Jaguar: Connection Procedures

To connect to Jaguar from a UNIX-based system type the following in your terminal:

`ssh userid@jaguarpf.ccs.ornl.gov` - Cray XT5

Enter PASSCODE: PIN + 6 digits from RSA® SecurID

Logging into Jaguar: One-Time Password (OTP) Authentication

RSA® SecurID - Quick Start Guide

All NCCS systems currently use OTPs as their authentication method. To login to NCCS systems, an RSA SecurID key fob is required!

Activating your SecurID key fob:

1. Return the completed NCCS token activation form to the address provided on the form. Once the form is received by NCCS, the RSA OTP token will be enabled, and you will be notified by email.
2. Initiate an SSH connection to home.ccs.ornl.gov
3. When prompted for a PASSCODE, enter the token code shown on the fob. You will be asked if you are ready to set your PIN. Answer with "Y."
4. You will then be prompted to enter a PIN. Enter a 4- to 6-digit number you can remember. Reenter your PIN when prompted.
5. You will be prompted to enter your full PASSCODE. To do so, wait until the next token code appears on your fob and enter your PASSCODE, which is now your PIN + the 6-digit token code displayed on your fob. For example, if the pin was 1111 and the token code was 223344, then the full PASSCODE would be 1111223344.
6. Your PIN is now set, and your fob is activated and ready for use. Log on using the procedure outlined in

Old design

Time bars

Token code

New design

Logging into Jaguar: Connection Options

X11 Tunneling

Automatic forwarding of the X11 display to a remote computer is highly recommended with the use of SSH and a local X server. To set up an automatic X11 tunneling with SSH, do one of the following:

1. Command line: Invoke ssh with the -X option:

```
ssh -X userid@jaguarpf.ccs.ornl.gov
```

   Note 1: use of the -x (lowercase x) option will disable X11 forwarding
   Note 2: use of the -Y option (instead of -X) is necessary on some systems to enable "trusted" X11 forwarding

2. Configuration file: Edit (or create) the .ssh/config file to have the following line in it:

```
ForwardX11 yes
```

3. Graphical Menu: Many SSH clients have a menu to change the configuration settings.
   PuTTY: check the box next to Connection --> SSH --> X11 --> Enable X11 Forwarding
   Note 1: Unix-like systems, with the exception of Mac OS-X, offer native X11 support. Apple does provide an implementation for OS-X, available from the Apple website.
   Note 2: For Windows systems you can also use free Xming software.
   Note 3: PuTTY stores configuration settings for each server separately.

Login Nodes

- When you login to Jaguar, you will be placed on a “login node”
- Login nodes are used for basic tasks such as file editing, code compilation, data backup, and job submission
- These nodes provide a full SUSE Linux environment, complete with compilers, tools, and libraries
- The login nodes should not be used to run production jobs. Production work should be performed on the systems compute resources.
- Serial jobs (post-processing, *etc*) may be run on the compute nodes as long as they are statically linked

Compute (Batch) Nodes

- All MPI/OpenMP user applications execute on batch or compute nodes
- Batch nodes provide **limited** Linux environment – Compute Node Linux (CNL)
- Compute nodes can see only the Lustre scratch directories
- Access to compute resources is managed by the PBS/TORQUE – batch system manager
- Job scheduling is handled by Moab, which interacts with PBS/TORQUE and the XT system software.

File Systems: User's Directories

Each user is provided the following space resources:

- Home directory - NFS Filesystem
  `/ccs/home/$USER`
- Work directory/Scratch space - Lustre Filesystem
  `/tmp/work/$USER`
- Project directory - NFS Filesystem
  `/ccs/proj/projectid`
- `Lustre Project Directory Filesystem`
`/tmp/proj/projectid`
- HPSS storage

File Systems: Basics

- The Network File Service (NFS) server contains user's home directories, project directories, and software directories .
- Compute nodes can only see the Lustre work space
  - The NFS-mounted home, project, and software directories are not accessible to the compute nodes.
- Shared Lustre area (SPIDER) is now available on compute nodes and is the only scratch area for the XT5.
- Executables must be executed from within the Lustre work space:
  - `/tmp/work/$USER` (XT4 and XT5)
- Batch jobs can be submitted from the home or work space. If submitted from a user's home area, a batch script should cd into the Lustre work space directory (`cd $PBS_O_WORKDIR`) prior to running the executable through aprun.
- All input/output for jobs on compute nodes must reside in the Lustre work space

File Systems: Home Directory

- Each user is provided a home directory to store frequently used items such as source code, binaries, and scripts. Home directories are located in a Network File Service (NFS) that is accessible from all NCCS resources.

- Home directory - NFS Filesystem

  Location: `/ccs/home/$USER`

- Accessible from all NCCS systems
- NFS does not provide the highest performance
- Default storage limit of 5 GB
- To find your quota and usage in NFS, use the `quota` command
- Regularly backed up

File Systems: Work Directory

- Work space is available on each NCCS high-performance computing (HPC) system for temporary files and for staging large files from and to the High Performance Storage System (HPSS). To ensure adequate work space is available for user's jobs, a script that finds and deletes old files runs on the system nightly. Thus, it is critical to archive files from the scratch area as soon as possible.
- Work directory/Scratch space - Lustre Filesystem

```
    /tmp/work/$USER
```

- The path `/tmp/work/$USER` is available on all NCCS HPC systems.

/tmp/work/$USER is Not backed up!

File Systems: Project Directory

- Each project is provided a directory shared by the project to store data such as source code, binaries, and scripts. Project directories are located in a Network File Service (NFS) that is accessible from all NCCS resources.

- Project directory - NFS Filesystem
  - Location: `/ccs/proj/projectid`
  - Accessible from all NCCS systems
  - Default storage limit of 10 GB
- Project directory - Lustre Filesystem
  - Location: `/tmp/proj/projectid`
  - Accessible from Jaguar, lens NCCS systems
  - Default storage limit of 1 TB
- By default, project directories are created with 770 permissions and the project ID group as the group owner.

Software Environment: `module` command

Loading Commands

- **`module [load||unload]`** ***`my_module`***
  - Loads/Unloads module *`my_module`*
  - e.g., `module load subversion`
- **`module swap`** ***`module#1 module#2`***
  - Replaces *`module#1`* with *`module#2`*
  - e.g., `module swap PrgEnv-pgi PrgEnv-gnu`

Informational Commands

- **`module help my_module`**
  - Lists available commands and usage
- **`module show my_module`**
  - Displays the actions upon loading `my_module`
- **`module list`**
  - Lists all loaded modules
- **`module avail [`*`name`*`]`**
  - Lists all modules [beginning with *`name`*]
  - e.g., `module avail gcc`

Compiling: System Compilers

The following compilers should be used to build codes on Jaguar:

Language	Compiler
C	`cc`
C++	`CC`
Fortran 77, 90 and 95	`ftn`

Note that cc, CC and ftn are actually the Cray XT Series wrappers for invoking the PGI, GNU or Pathscale compilers (discussed later…)

Compiling: Default Compilers

- Default compiler is PGI. The list of all packages is obtained by
  - `module avail PrgEnv`
- To use the Cray wrappers with other compilers the programming environment modules need to be swapped, i.e.
  - `module swap PrgEnv-pgi PrgEnv-gnu`
  - `module swap PrgEnv-pgi PrgEnv-cray`
- To just use the GNU/Cray compilers directly load the GNU/Cray module you want:
  - `module load PrgEnv-gnu/2.1.50HD`
  - `module load PrgEnv-cray/1.0.1`
- It is possible to use the GNU compiler versions directly without loading the Cray Programming Environments, but note that the Cray wrappers will probably not work as expected if you do that.

Compiling: Useful Compiler Flags (PGI)

General:

Flag	Comments
`-mp=nonuma`	Compile multithreaded code using OpenMP directives

Debugging:

Flag	Comments
`-g`	For debugging symbols; put first
`-Ktrap=fp`	Trap floating point exceptions
`-Mchkptr`	Checks for unintended dereferencing of null pointers

Optimization:

Flag	Comments
`-fast`	Equivalent to **`-Mvect=sse -Mscalarsse -Mcache_align -Mflushz`**
`-fastsse`	Same as **`-fast`**
`-Mcache_align`	Makes certain that arrays are on cache line boundaries
`-Munroll=c:`*n*	Unrolls loops ***n*** times (e.g., ***n***=4)
`-Mipa=fast,inline`	Enables interprocedural analysis (IPA) and inlining, benefits for C++ and Fortran
`-Mconcur`	Automatic parallelization

Running Jobs: Introduction

- When you log into Jaguar, you are placed on one of the login nodes.
- Login nodes should be used for basic tasks such as file editing, code compilation, data backup, and job submission.
- The login nodes **should not be** used to run production jobs. Production work should be performed on the system's compute resources.
- On Jaguar, access to compute resources is managed by the PBS/TORQUE. Job scheduling and queue management is handled by Moab which interacts with PBS/TORQUE and the XT system software.
- Users either submit the job scripts for batch jobs, or submit a request for interactive job.
- The following pages provide information for getting started with the batch facilities of PBS/TORQUE with Moab as well as basic job execution.

Running Jobs: Glossary

- PBS/TORQUE is an open source resource manager providing control over batch jobs and distributed compute nodes. It is a community effort based on the original PBS project.
- Portable Batch System (or simply PBS) is the computer software that performs job scheduling. Its primary task is to allocate computational tasks, i.e., batch jobs, among the available computing resources. PBS is supported as a job scheduler mechanism by Moab.
- Batch jobs are set up so they can be run to completion without human interaction, so all input data is preselected through scripts or command-line parameters. This is in contrast to "online" or interactive programs which prompt the user for such input.

Running Jobs: Batch Scripts

- Batch scripts can be used to run a set of commands on a systems compute partition.
- The batch script is a shell script containing PBS flags and commands to be interpreted by a shell.
- Batch scripts are submitted to the batch manager, PBS, where they are parsed. Based on the parsed data, PBS places the script in the queue as a job.
- Once the job makes its way through the queue, the script will be executed on the head node of the allocated resources.

Running Jobs: Example Batch Script

```
1: #!/bin/bash
2: #PBS -A XXXYYY
3: #PBS -N test
4: #PBS -j oe
5: #PBS -l walltime=1:00:00,size=192
6:
7: cd $PBS_O_WORKDIR
8: date
9: aprun -n 192 ./a.out
```

NOTE: Since users cannot share nodes, size requests must be a multiple of 12 on the XT5.

This batch script can be broken down into the following sections:

- Shell interpreter
  - Line 1
  - Can be used to specify an interpreting shell.
- PBS commands
  - The PBS options will be read and used by PBS upon submission.
  - Lines 2–5
    - 2: The job will be charged to the XXXYYY project.
    - 3: The job will be named “test.”
    - 4: The jobs standard output and error will be combined.
    - 5: The job will request 192 cores for 1 hour.
  - Please see the PBS Options page for more options.
- Shell commands
  - Once the requested resources have been allocated, the shell commands will be executed on the allocated nodes head node.
  - Lines 6–9
    - 6: This line is left blank, so it will be ignored.
    - 7: This command will change directory into the script's submission directory. We assume here that the job was submitted from a directory in /lustre/scratch/.
    - 8: This command will run the date command.
    - 9: This command will run the executable a.out on 192 cores with a.out.

Running Jobs: Submitting Batch Jobs - `qsub`

- All job resource management handled by Torque.
- Batch scripts can be submitted for execution using the `qsub` command.
- For example, the following will submit the batch script named `test.pbs`:

```
qsub test.pbs
```

- If successfully submitted, a PBS job ID will be returned. This ID can be used to track the job.

Running Jobs: Interactive Batch Jobs

- Batch scripts are useful for submitting a group of commands, allowing them to run through the queue, then viewing the results. It is also often useful to run a job interactively. However, users are not allowed to directly run on compute resources from the login module. Instead, users must use a batch-interactive PBS job. This is done by using the `-I` option to `qsub`.
- For interactive batch jobs, PBS options are passed through `qsub` on the command line:

```
qsub -I -A XXXYYY -q debug -V -l size=24,walltime=1:00:00
```

This request will…

`-I`	Start an interactive session
`-A`	Charge to the "XXXYYY" project
`-q debug`	Run in the debug queue
`-V`	Import the submitting users environment
`-l size=24,walltime=1:00:00`	Request 24 compute cores for one hour

Running Jobs: PBS Options

Necessary PBS options:

Option	Use	Description
`A`	`#PBS -A <account>`	Causes the job time to be charged to <account>. The account string XXXYYY is typically composed of three letters followed by three digits and optionally followed by a subproject identifier. The utility showusage can be used to list your valid assigned project ID(s). This is the only option required by all jobs.
`l`	`#PBS -l size=<cores>`	Maximum number of compute cores. Must request an entire node (multiples of 4 on the XT4, and 12 on the XT5).
	`#PBS -l walltime=<time>`	Maximum wall-clock time. <time> is in the format HH:MM:SS. Default is 45 minutes.

Running Jobs: PBS Options (cont.)

Commonly used, but not necessary PBS Options:

Option	Use	Description
`l`	`#PBS -l feature=<target>`	Run only on the specified target. Currently the available target is XT5 with 1 or 2 GB of memory per node. The default is to run on the first available. It is recommended to use the default. The other option is to specify "2gbpercore" to run on 16 GB nodes only.
`o`	`#PBS -o <name>`	Writes standard output to <name> instead of <job script>.o$PBS_JOBID. $PBS_JOBID is an environment variable created by PBS that contains the PBS job identifier.
`e`	`#PBS -e <name>`	Writes standard error to <name> instead of <job script>.e$PBS_JOBID.
`j`	`#PBS -j {oe,eo}`	Combines standard output and standard error into the standard error file (eo) or the standard out file (oe).
`m`	`#PBS -m a`	Sends email to the submitter when the job aborts.
	`#PBS -m b`	Sends email to the submitter when the job begins.
	`#PBS -m e`	Sends email to the submitter when the job ends.
`M`	`#PBS -M <address>`	Specifies email address to use for -m options.
`N`	`#PBS -N <name>`	Sets the job name to <name> instead of the name of the job script.
`S`	`#PBS -S <shell>`	Sets the shell to interpret the job script.
`q`	`#PBS -q <queue>`	Directs the job to the specified queue.This option is not required to run in the general production queue.
`V`	`#PBS -V`	Exports all environment variables from the submitting shell into the batch shell.

Running Jobs: PBS Environment Variables

- `PBS_O_WORKDIR`
  - PBS sets the environment variable PBS_O_WORKDIR to the directory where the batch job was submitted.
  - By default, a job starts in your home directory.
  - Include the following command in your script if you want it to start in the submission directory:

```
cd $PBS_O_WORKDIR
```

- `PBS_JOBID`
  - PBS sets the environment variable PBS_JOBID to the job's ID.
  - A common use for PBS_JOBID is to append the job's ID to the standard output and error file(s), such as the following:

```
PBS -o scriptname.o$PBS_JOBID
```

- `PBS_NNODES`
  - PBS sets the environment variable PBS_NNODES to the number of cores requested. This means that number of nodes requested on a 12-core architecture would be $PBS_NNODES/12.

Running Jobs: Altering Batch Jobs – `qdel,qhold,qrls`

- Command: `qdel`
  - Jobs in the queue in any state can be stopped and removed from the queue using the command qdel.
  - For example, to remove a job with a PBS ID of 1234, use the following command: `qdel 1234`

- Command: `qhold`
  - Jobs in the queue in a non-running state may be placed on hold using the qhold command. Jobs placed on hold will not be removed from the queue, but they will not be eligible for execution.
  - For example, to move a currently queued job with a PBS ID of 1234 to a hold state, use the following command: `qhold 1234`

- Command: `qrls`
  - Once on hold the job will not be eligible to run until it is released to return to a queued state. The qrls command can be used to remove a job from the held state.
  - For example, to release job 1234 from a held state, use the following command: `qrls 1234`

Running Jobs: Monitoring Job Status - `qstat`

PBS and Moab provide multiple tools to view queue, system, and job statuses.
Command: `qstat`
Use `qstat -a` to check the status of submitted jobs:

```
nid00004:
                                                       Req'd  Req'd    Elap
Job ID Username Queue Jobname SessID NDS Tasks Memory Time   S Time
------ -------- ----- ------- ------ --- ----- ------ -----  - -----
29668   user1   batch job2    21909   1  256   --     08:00 R 02:28
29894   user2   batch run128   --     1  128   --     02:30 Q --
29895   user3   batch STDIN   15921   1  1     --     01:00 R 00:10
29896   user2   batch jobL    21988   1  2048  --     01:00 R 00:09
29897   user4   debug STDIN   22367   1  2     --     00:30 R 00:06
29898   user1   batch job1    25188   1  1     --     01:10 C 00:00
```

Job ID	PBS assigned job ID.
Username	Submitting user's user ID.
Queue	Queue into which the job has been submitted.
Jobname	PBS job name. This is given by the PBS -n option in the PBS batch script. Or, if the -n option is not used, PBS will use the name of the batch script.
SessID	Associated session ID.
NDS	PBS node count. Not accurate; will be one.
Tasks	Number of cores requested by the job's -size option.
Req'd Memory	Job's requested memory.
Req'd Time	Job's given wall time.
S	Job's current status. See the status listings below.
Elap Time	Job's time spent in a running status. If a job is not currently or has not been in a run state, the field will be blank.

Status Value	**Meaning**
E	Exiting after having run
H	Held
Q	Queued; eligible to run
R	Running
S	Suspended
T	Being moved to new location
W	Waiting for its execution time
C	Recently completed (within the last 5 minutes)

Running Jobs: `showq, checkjob`

Command : `showq`
The Moab utility `showq` gives a more detailed description of the queue and displays it in the following states:

Active	These jobs are currently running.
Eligible	These jobs are currently queued awaiting resources. A user is allowed five jobs in the eligible state.
Blocked	These jobs are currently queued but are not eligible to run. Common reasons for jobs in this state are jobs on hold, the owning user currently having five jobs in the eligible state, and running jobs in the longsmall queue.

Command : `checkjob`
The Moab utility `checkjob` can be used to view details of a job in the queue.
For example, if job 736 is a job currently in the queue in a blocked state, the following can be used to view why the job is in a blocked state:
`checkjob 736` The return may contain a line similar to the following:

```
BlockMsg: job 736 violates idle HARD MAXJOB limit of 2 for
      user (Req: 1 In Use: 2)
```

This line indicates the job is in the blocked state because the owning user has reached the limit of two jobs currently in the eligible state.

Running Jobs: `showstart, showbf, xtprocadmin`

Command : `showstart`

The Moab utility `showstart` gives an estimate of when the job will start.

```
showstart 100315
job 100315 requires 16384 procs for 00:40:00
Estimated Rsv based start in 15:26:41 on Fri Sep 26 23:41:12
Estimated Rsv based completion in 16:06:41 on Sat Sep 27 00:21:12
```

Since the start time may change dramatically as new jobs with higher priority are submitted, so you need to periodically rerun the command.

Command : `showbf`

The Moab utility `showbf` gives the current backfill. This can help to build a job which can be backfilled immediately. As such, it is primarily useful for small jobs.

Command : `xtprocadmin`

The utility `xtprocadmin` can be used to see what jobs are currently running and which nodes they are running on.

Running Jobs: Job Execution - `aprun`

- By default, commands will be executed on the job's associated service node.
- The `aprun` command is used to execute a job on one or more compute nodes.
- The XT's layout should be kept in mind when running a job using `aprun`. The XT5 partition currently contains two hex-core processors (a total of 12 cores) per compute node. While the XT4 partition currently contains one quad-core processor (a total of 4 cores) per compute node.
- The `PBS size` option requests compute cores.

Running Jobs: Basic `aprun` options

Option	Description
`-D`	Debug (shows the layout aprun will use)
`-n`	Number of MPI tasks. Note: If you do not specify the number of tasks to aprun, the system will default to 1.
`-N`	Number of tasks per Node. (XT5: 1 – 12) NOTE: Recall that the XT5 has two Opterons per compute node. On the XT5, to place one task per Opteron, use -S 1
`-m`	Memory required per task.
`-d`	Number of threads per MPI task. Note: As of CLE 2.1, this option is very important. If you specify OMP_NUM_THREADS but do not give a -d option, aprun will allocate your threads to a single core. You must use OMP_NUM_THREADS to specify the number of threads per MPI task, and you must use -d to tell aprun how to place those threads.
`-S`	Number of PEs to allocate per NUMA node.
`-ss`	Strict memory containment per NUMA node.

Running Jobs: XT5 example

`aprun –n 24 ./a.out` will run a.out across 24 cores. This requires two compute nodes. The MPI task layout would be as follows:

Compute Node 1

Opteron 0						**Opteron 1**					
Core 0	**Core 1**	**Core 2**	**Core 3**	**Core 4**	**Core 5**	**Core 0**	**Core 1**	**Core 2**	**Core 3**	**Core 4**	**Core 5**
0	1	2	3	4	5	6	7	8	9	10	11

Compute Node 2

Opteron 0						**Opteron 1**					
Core 0	**Core 1**	**Core 2**	**Core 3**	**Core 4**	**Core 5**	**Core 0**	**Core 1**	**Core 2**	**Core 3**	**Core 4**	**Core 5**
12	13	14	15	16	17	18	19	20	21	22	23

The following will place tasks in a round robin fashion.

```
> setenv MPICH_RANK_REORDER_METHOD 0
> aprun -n 24 a.out
```

```
Rank 0,  Node 1, Opteron 0, Core 0
Rank 1,  Node 2, Opteron 0, Core 0
Rank 2,  Node 1, Opteron 0, Core 1
Rank 3,  Node 2, Opteron 0, Core 1
Rank 4,  Node 1, Opteron 0, Core 2
Rank 5,  Node 2, Opteron 0, Core 2
Rank 6,  Node 1, Opteron 0, Core 3
Rank 7,  Node 2, Opteron 0, Core 3
Rank 8,  Node 1, Opteron 0, Core 4
Rank 9,  Node 2, Opteron 0, Core 4
Rank 10, Node 1, Opteron 0, Core 5
Rank 11, Node 2, Opteron 0, Core 5
```

```
Rank 0,  Node 1, Opteron 1, Core 0
Rank 1,  Node 2, Opteron 1, Core 0
Rank 2,  Node 1, Opteron 1, Core 1
Rank 3,  Node 2, Opteron 1, Core 1
Rank 4,  Node 1, Opteron 1, Core 2
Rank 5,  Node 2, Opteron 1, Core 2
Rank 6,  Node 1, Opteron 1, Core 3
Rank 7,  Node 2, Opteron 1, Core 3
Rank 8,  Node 1, Opteron 1, Core 4
Rank 9,  Node 2, Opteron 1, Core 4
Rank 10, Node 1, Opteron 1, Core 5
Rank 11, Node 2, Opteron 1, Core 5
```

Third-Party Software

- NCCS has installed many third-party software packages, libraries, etc., and created module files for them
  - Third-party applications (e.g., MATLAB, GAMESS)
  - Latest versions or old versions not supported by vendor
  - Suboptimal versions to do proof-of-concept work (e.g., blas/ref)
  - Debug versions
- NCCS modules available via `module load` command, installed in `/sw/xt/` directory

File Systems: High Performance Storage System (HPSS)

- HPSS is an archival Back-up system which consists of
  - two types of storage technology:
    - disk – "on-line" for frequently/recently accessed files
    - tape – "off-line" for very large or infrequently accessed files
  - Linux servers
  - High Performance Storage System software
- Tape storage is provided by robotic tape libraries.
- HPSS has three SL8500 tape libraries. Each can hold up to 10,000 cartridges.
- The StorageTek SL8500 libraries house a total of
  - twenty-four T10000A tape drives (500 gigabyte cartridges, uncompressed)
  - thirty-six T10000B tape drives (1 terabyte cartridges, uncompressed).
- Each drive has a bandwidth of 120 MB/s

File Systems: Using `hsi` and `htar` on HPSS

- Each user of an NCCS system is provided an account on the HPSS. The user's login name for HPSS is the same as for all other NCCS systems. Authorization to HPSS is by means of the user's SecurID token.
- **Users are encouraged to use `hsi` when dealing with a small number of files, and `htar` for large numbers of files.**
- The `hsi` utility provides the ability to access and transfer data to and from the NCCS HPSS for both disk and tape file systems. Issuing the command `hsi` will start HSI in interactive mode.
- Information on HSI may be found from the NCCS systems through the command
  - `hsi help`
- The `htar` command – works like Unix "tar"
- Below is an example of storing and getting a bunch of files in a directory using tar and HSI. HSI can read from standard input and write to standard output:
  - `tar cvf - . | hsi put - : <filename.tar>`
  - `hsi get - : <filename.tar> | tar xvf -`